Time-, Energy-, and Monetary Cost-Aware Query[*] Processing in a Mobile-Cloud Database Environment

Jonathan Mullen[1] Mikael Perrin[1] Le Gruenwald[1] Laurent d'Orazio[2]

[1] School of Computer Science
University of Oklahoma
Norman, Oklahoma, USA
{jonathan,mikael.perrin,ggruenwald}@ou.edu

[2] CNRS, UMR 6158, LIMOS
Blaise Pascal University
Clermont-Ferrand, France
laurent.dorazio@isima.fr

ABSTRACT
Growing demand for more mobile access to data is only matched by the growth of large and complex data. The availability and scalability of cloud resources when combined with techniques of caching and distributed computation provide tools to address these problems, but bring up new multi-dimensional optimization challenges such as execution time, monetary cost, and power consumption. The plethora of various cloud providers with varying pricing schemes only further complicates the user's problem. To address these issues we present a three-tier model that formalizes the query planning and execution between mobile users, data owners and cloud providers, allowing state holders to impose constraints on time, money and energy consumption and understand the possible tradeoffs between them.

1. INTRODUCTION
Mobility, while not a new concept, has increasingly been the focus of both academic and industry research. The allure of mobile computing is that it provides the users access to computational resources independent of their location. While the applications and possibilities of ubiquitous access to computing resources without having to be tethered to a desk are still being explored, the limitations are fairly well known and force us to reexamine problems long considered solved in the non-mobile case. For example, mobile devices commonly face issues of constrained resources such as energy, processing power, storage capacity, available memory, network communication and limited display and input methods. The disproportionate capabilities of mobile devices, when compared to traditional servers, are quite stark; the divide becomes even more apparent when one considers the capabilities of clustered cloud resources. To address this disparity between capabilities, we have largely fallen back onto the client-server model, where the mobile device acts as a client traveling with the user querying the more capable cloud servers for complex and intensive needs.

While the simple client-server model is well understood it does not fully address the advantages such as elasticity, or the challenges such as two-dimensional optimization with respect to time and money, heterogeneous pricing models, or heterogeneous capabilities. These challenges often make it hard for users of cloud services to understand the cost associated with query, or the possible tradeoff between cost and performance.

For example, consider a doctor who wants to retrieve all the patient information on his/her mobile device. The patient data is spread among several databases, some of which may reside on the data-owners hardware (database servers owned by the organization providing the service), while others could be spread across various cloud providers. Furthermore, the compilation of this data can be computation and data intensive. The problem is further complicated by the fact that in addition to considering the constraints of client and server resources, one must also consider the elastic capabilities and pricing of cloud services.

The goal of this paper is to provide our vision for designing a system that not only effectively utilizes the unique capabilities of mobile and cloud resources by allowing for one to easily take advantage of the elastic nature of the cloud in order to retrieve data, but also to respect the constraints of time, energy on the mobile device and monetary cost to use the different cloud services.

The rest of the paper is organized into the following sections. In Section 2, we present our proposed three-tier architecture. In Sections 3 and 4, we discuss our ideas and associated research challenges on caching and query cost estimation strategies on the mobile devices and query cost optimization strategies on clouds to reduce the amount of time, energy and money necessary for a given query. Finally in Section 5, we provide our conclusions.

2. CLIENT-SERVER-CLOUD ARCHITECTURE
To address the problems of elastic execution, cost estimation, data locality common in interactions that involve mobile users querying data owners who utilize cloud service providers, we propose a three-tier architecture as described in Figure 1 - Proposed Architecture. The first tier, or mobile devices, represents the end user where queries are posed and results are expected. While mobility is not inherently part of our architecture, the assumption is that end user devices will be resource constrained (energy, processing power and bandwidth). The second tier, or data owner, represents statically available non-elastic resources controlled by the owner of the data on which the query is being posed. Lastly, the third tier is the cloud

[*] This work is partially supported by the National Science Foundation Award No. 1349285

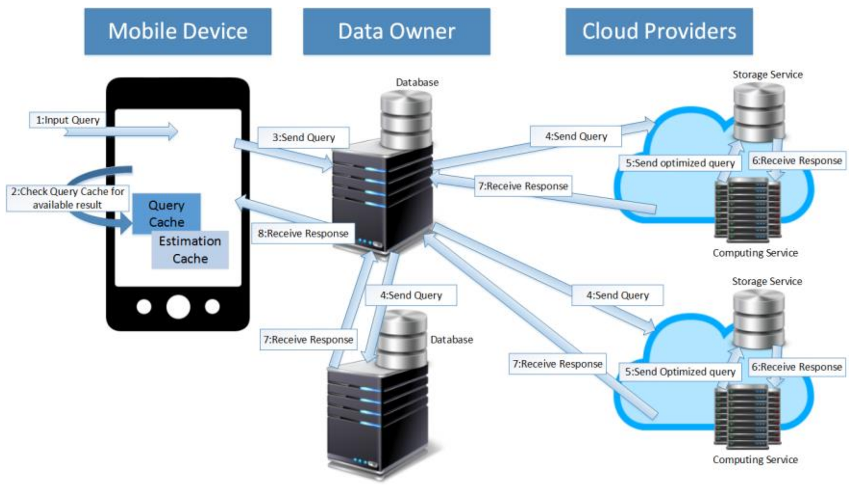


Figure 1 - Proposed Architecture

services tier comprised of elastic cloud resources, to which the data owner has access.

For illustration, here we use a hospital application as an example of the three-tier architecture. Imagine a hospital director needing to look up doctor and project information using a tablet in a meeting; this would be the mobile tier. The hospital likely has some existing server infrastructure with physical limitations. These resources are not elastic; adding more server capacity would require both upfront capital for a new server in addition to any assembly, shipping and configuration lead time. These static hospital resources would be the data owner tier. To address burst computation needs, the hospital utilizes various IaaS, PaaS and SaaS [1] as dictated by their demands. Unlike the constrained data owner resources, these cloud resources are elastic in nature, meaning they can easily be scaled up, or down as required. Usually this will involve some sort of pricing scheme based on the resources utilized; this is the cloud resources tier.

On the mobile device, a well-known technique used to reduce the number of interactions with a server is called semantic caching [2]. Semantic caching involves not only storing the results from previous queries, but also maintaining the metadata associated with each query. With this metadata, further server communication is either eliminated when the result of a query is stored entirely in cache, or reduced when the cache contains only a part of the data required by the query. To determine this, the technique of query trimming like the one described in [3] is used to compare the input query with the query contained in each metadata entry. In the case when the cache partially contains the query result, a probe query will be created to retrieve the existing data from the mobile cache, and a remainder query will be created to retrieve the data not contained in the mobile cache. The remainder query will be sent to the data owner for execution.

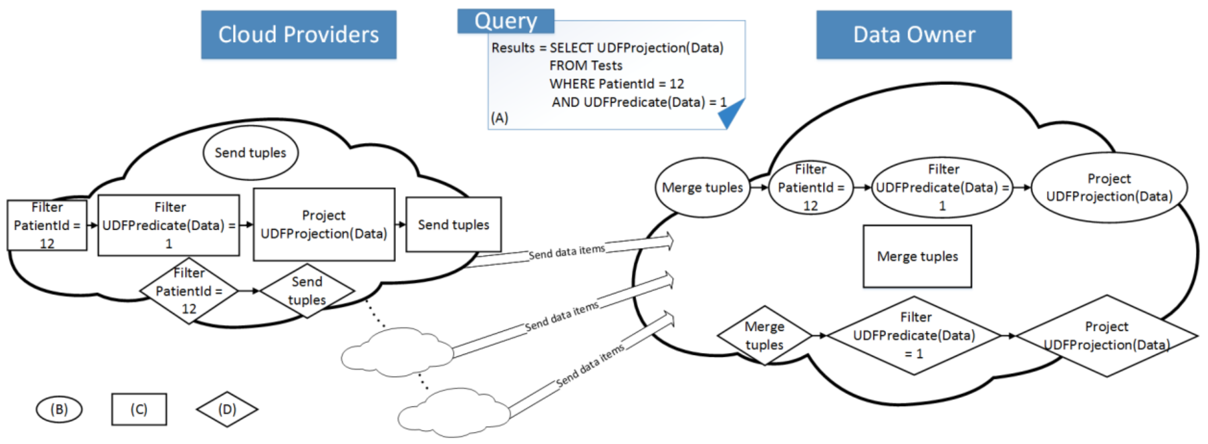
This query interaction, be it a full query, a remainder query, or a prefetched query, between the mobile device and the data owner constitutes the primary interaction between the two macro components of our solution. The mobile component is responsible for maintaining a local cache and deciding what query to evaluate. The data-owner and cloud component, in addition to providing execution estimation to the mobile component for use in query planning, is also responsible for evaluating the mobile query while efficiently utilizing the available cloud resources. Just as the mobile component must consider optimizing for both time and energy, the cloud component must consider optimizing for time and budget. With this comes the concept of elasticity. Elasticity defines the effect that additional resources or money has on the completion time of a query. For example, it may be preferable to focus budget resources on executing a highly elastic query where 50% increase in budget decreases execution time by 200%, rather than a weekly elastic query where decreasing the budget by 50% only decreases completion time by 10%.

While many works have studied cloud data management and query execution, the scope of investigation is usually limited in the sense that only execution time is optimized [4]; or compute resources or cloud-pricing models are considered homogeneous [5]. By extending the previous works on these disjoint problems we present a single solution that considers time and budget across heterogeneous clouds, with heterogeneous pricing models, allowing both mobile user and data-owner to access their data while providing fine grain control of execution time and cost.

3. MOBILE CACHE AND ESTIMATION

Introducing semantic caching makes it mandatory to find a trade-off between running the query on the cloud and running the query on the mobile with respect to efficiency, money and energy consumption. In particular, it is necessary to estimate the amounts of time and energy to be spent to retrieve data on the mobile device and to compare them with the time and energy to retrieve data on the cloud. In some cases, these amounts of time and energy consumed can be bigger than those incurred for processing the whole query on the cloud [6]. Therefore, it is important to estimate the costs in terms of time, energy and money for each query.

We envision that there are two cache structures on a mobile device: query cache and estimation cache. The query cache may be based on semantic caching which can be implemented using the definitions, structures and algorithms given in existing works, such as [2], [3], [7], [8], [9], [10]. With this, an input query may be answered using some available cache entries and therefore, only the missing data will be requested from the cloud. For example, if the input query has the predicate "*Patient.Age < 28*" and one entry in the query cache corresponds to the predicate "*Patient.Age < 29*", where Patient is a relation in the database and Age is an attribute in this relation, then it is possible to answer the input query thanks to the content of the query cache. In a semantic cache, each entry contains the information items on



```
Results = SELECT UDFProjection(Data)
          FROM Tests
          WHERE PatientId = 12
           AND UDFPredicate(Data) = 1
```

Figure 2 - Query (A), Operator Graph (B - D), Possible Execution Plans

the corresponding queries (relation, attributes, time stamp, predicates, etc.) and a semantic region where the result tuples are stored.

For each cache entry, it is necessary to check that the relation and the attributes of the segment in the query cache are the same as those of the given query. Also, the query predicates has to be analyzed in order to know if the query cache content answers the input query completely, partially, or not at all. Once we know which entries can be used to answer the query on the mobile device, we need to process each semantic region to retrieve the wanted tuples. The time and energy to retrieve those data depend on the structure used to store the tuples in each region. However, if the query is processed on the mobile device, no money is paid to the cloud provider to retrieve the data. Consequently, it is important to ask the cloud to provide an estimation of how much time and money it will take to process the query on its resources. With this estimation, the mobile device will then estimate the energy it will consume to process the query on those cloud services.

Since computing this estimation requires time, energy and money, the mobile device can be equipped with an estimation cache to reduce such overheads. Each estimation cache entry should contain the time to process the query in seconds, the energy to process the query in Joules, and the money cost to process the query in dollars. Once we have analyzed the mobile device's cache and we have asked the cloud to provide the cost of processing such a query on its services, two estimations are available: one to run the query on the mobile device, and one to run the query on the cloud. Those estimations are then used to check whether or not the given constraints on remaining battery life, money and time can be met. Following this result, the query plan is built in order to know where to process the different queries (on the mobile device or on the cloud).

4. CLOUD QUERY EXECUTION AND COST ESTIMATION

There is a broad body of work that has been done in optimizing cloud query execution, also known as data flow execution [5], [4] [11], [12]. While the majority of work has been focused on optimizing completion time, with the advent of on-demand computing, monetary cost optimization has also been considered [5]. No longer must execution resources be algorithmically limited as a given, but rather can be scaled according to the query at hand, and the constraining budget. A highly elastic query implies that a little additional money has a large effect on execution time. Such a metric is critical to cost-benefit analysis, as it allows the user to make informed decisions regarding resources.

Given the scope and size of the problem, abstraction layers are often introduced to help reason about the problem. The principal example of this is the Map-Reduce model [5] which has proven to be extremely adaptable, although not always trivial to apply to a problem. This is evidenced by the large number of high-level languages, such as Pig [11] and Hive [13], built on top of the Map-Reduce model.

One common theme is the emergence of directed-acyclic graphs (DAGs) as a means for defining and reasoning about cloud queries or data flows [4], [5], [14], [12]. In this type of formulation, nodes represent operations or calculations, and edges represent the flow of data between nodes. Depending of the specific formulation, nodes either represent logical operations in a plan, or concrete operations to be performed by specific nodes on partitions of data. The former is referred to as an operator graph, while the latter is called a concrete operator graph.

Given a concrete operator graph, proper statistics about operator execution and data cardinality, it is trivial to estimate a schedule and thus the cost of execution for a given pricing model. In many ways, this abstraction has reduced the problem of cloud query execution and cost estimation to a form effectively equivalent to RDBMS execution and cost estimation. The primary difference compared to the RDBMS case is that the size of the possible solution space is much larger given the scalable nature or resources.

Previous solutions have addressed the large search space of possible plans through a nested loop approach [5]. This naïvely serializes the optimization process by trying to optimally schedule a concrete operator graph given an optimal operator graph. The outer loop will generate possible resource limits, and the inner loop optimally creates a concrete operator graph for the given resource limits, and desired constraints. The process itself is fairly generic, and can be tweaked for different purposes by adjusting the bounds on the available resource search space, the optimal stopping criteria, and most importantly, the inner optimizer.

Our proposal is to expand and ensemble existing disjointed solutions into a unified system for estimating the cost, and executing queries defined as DAGs across heterogeneous clouds.

In particular we aim to not only consider pricing models in regards to execution time quantum, but other cloud pricing features as well, such as data storage and bandwidth.

Consider the case of operating on the result of joining two relations A and B, where A and B are initially located on different clouds. If bandwidth between the clouds is sufficient, and bandwidth cost is ignored or is treated uniformly, one might generate a plan where nodes intercommunicate between clouds. This might even be an optimal plan when just considering execution time. By adding in the consideration of bandwidth cost, we have significantly changed the calculus of the situation.

This is specifically true when we consider the wide variation in bandwidth pricing between cloud providers. For example consider the differences between Amazon EC2 and Rackspace. While EC2 charges for almost all forms of traffic coming from a node, Rackspace offers an internal network where we can communicate with other nodes free of charge [15].

Empowered by a canonical form for cloud pricing models at a finer granularity, our proposed system might explore a plan counter-intuitive to time, where an initial time cost of transferring a relation between clouds leads to significant monetary savings over time as a result of decreased bandwidth costs overall.

For example, consider the situation where a relation *Tests* is stripped across several cloud providers, depending on the type of test, and where it was conducted. A hospital is considered the data-owner of its own tests, and thus the compilation of *Tests* for a patient may be achieved by many different execution plans depending on performance and privacy concerns. In Figure 2, we present those possible execution plans for a given query (Part A in Figure 2. The first possibility (Part B in Figure 2) is to get the tuples contained in the cloud providers' database corresponding to the *Tests* relation and send those to the data-owner where the tuple merge, the filtering and the projection will be processed. The second possibility (Part C in Figure 2) is to do all the filtering and the projection on every cloud provider and send the tuples to the data-owner where they will be merged. The third possibility (Part D in Figure 2) is to do part of the filtering on each cloud provider, send the tuples to the data-owner to merger the results, and then, process the remaining filtering and projection.

5. CONCLUSION AND OUTLOOK

This vision paper presents a proposed architecture to be able to optimize the time, energy and money within a Mobile-Cloud environment. Determining which query plan should be used to know whether to process the query on the mobile device, the data owner or the different cloud services is an issue that our approach is attempting to solve. Given our proposed three-tiered approach, the issue is quite large to consider as a single monolithic component. Instead we have chosen to look at our approach as an ensemble of two more manageable components. The first component looks at semantic caching and defines how we can estimate the time and energy needed to analyze the cache (query trimming) as well as the time and energy required to process the input query on it. The goal of such estimation is to be compared with the estimation time, energy and money returned by the cloud service. Only then can we make an intelligent decision on a query execution plan. In the second component, we present a set of techniques to both estimate and optimize the cost of query execution in the cloud with respect to both time and money. Specifically we are interested in exploiting the various capabilities and pricing models available for IaaS and SaaS resources. With this architecture, we can now allow the user to access data quickly without draining their battery, while empowering the data owner to impose response time, energy consumption and monetary cost constraints on user queries.

6. REFERENCES

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," *NIST*, vol. 53.6, p. 50, 2009.

[2] S. Dar, M. J. Franklin, B. T. Jonsson, D. Srivastava and M. Tan, "Semantic data caching and replacement," *VLDB*, vol. 96, pp. 330-341, 1996.

[3] Q. Ren, M. H. Dunham and V. Kumar, "Semantic caching and query processing," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 15.1, pp. 192-210, 2003.

[4] N. Bruno, S. Jain and J. Zhou, "Continuous cloud-scale query optimization and processing," in *Proceedings of the VLDB Endowment*, 2013.

[5] H. Kllapi, E. Sitaridi, M. M. Tsangaris and Y. Ioannidis, "Schedule optimization for data processing flows on the cloud," in *ACM SIGMOD International Conference on Management of data*, 2011.

[6] A. Carroll and G. Heiser, "An Analysis of Power Consumption in a Smartphone," in *USENIX annual technical conference*, 2010.

[7] B. Þ. Jónsson, M. Arinbjarnar, B. Þórsson, M. J. Franklin and D. Srivastava, "Performance and overhead of semantic cache management," *ACM Transactions on Internet Technology (TOIT)*, vol. 6.3, pp. 302-331, 2006.

[8] P. Godfrey and J. Gryz, "Answering queries by semantic caches," *Database and Expert Systems Applications*, vol. Springer Berlin Heidelberg, pp. 485-498, 1999.

[9] K. C. Lee, H. V. Leong and A. Si, "Semantic Query Caching in a Mobile Environment," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 3.2, pp. 28-36, 1999.

[10] B. Chidlovskii and U. M. Borghoff, "Semantic caching of Web queries," *VLDB*, vol. 9.1, pp. 2-17, 2000.

[11] C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *ACM SIGMOD international conference on Management of data*, 2008.

[12] R. Chaiken, B. Jenkins, P.-Å. Larson, B. Ramsey, D. Shakib, S. Weaver and J. Zhou, "SCOPE: easy and efficient parallel processing of massive data sets," in *Proceedings of the VLDB Endowment*, 2008.

[13] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," in *Proceedings of the VLDB Endowment*, 2009.

[14] K. Morton, M. Balazinska and D. Grossman, "ParaTimer: a progress indicator for MapReduce DAGs," in *ACM SIGMOD International Conference on Management of data*, 2010.

[15] "Rackspace, the open cloud compagny," Rackspace, US inc, [Online]. Available: http://www.rackspace.com. [Accessed May 2014].